

Exhibit B

SYNCHRONIZER.H

```

5  #if
    !defined(AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E
    0__INCLUDED_)
    #define
    AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E0__INC
    LUDED_

    // SYNCHRONIZER.H - Header file for your Internet Server
10  // Synchronizer Extension

    #include "resource.h"

    class isapithread;

15  class CSynchronizerExtension : public CHttpServer
    {
    protected:
        isapithread * m_pisapithread;
20  public:
        CSynchronizerExtension();
        ~CSynchronizerExtension();

    // Overrides
25  // Class Wizard generated virtual function overrides
        // NOTE - the Class Wizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
        //{AFX_VIRTUAL(CSynchronizerExtension)
        public:
30  virtual BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
        //}AFX_VIRTUAL
        virtual BOOL TerminateExtension(DWORD dwFlags);

        // TODO: Add handlers for your commands here.
35  // For example:

        void Default(CHttpServerContext* pCtxt);

        DECLARE_PARSE_MAP()

40  void BroadcastVideoEvents(CHttpServerContext* pCtxt, LPCTSTR lpUserName,
        LPCTSTR lpDiskId, long nLocationID, LPCTSTR lpBCA, LPCTSTR lpProcessorID,
        LPCTSTR lpDecoderID);
        void ServerInfo(CHttpServerContext* pCtxt, LPCTSTR lpDiskId, LPCTSTR
45  nLocationID);

```

```
//{{AFX_MSG(CSynchronizerExtension)
//}}AFX_MSG
};
```

5

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
```

10

```
#endif //
!defined(AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E
0__INCLUDED)
```

SYNCHRONIZER.CPP:

```

// SYNCHRONIZER.CPP - Implementation file for your Internet Server
// Synchronizer Extension
5
#include "stdafx.h"
#include "Synchronizer.h"
#include "isapithread.h"
#include "cominit.h"
10 #include <string>

////////////////////////////////////
// The one and only CWinApp object
// NOTE: You may remove this object if you alter your project to no
15 // longer use MFC in a DLL.

CWinApp theApp;

////////////////////////////////////
20 // command-parsing map

BEGIN_PARSE_MAP(CSynchronizerExtension, CHttpServer)
    // TODO: insert your ON_PARSE_COMMAND() and
    // ON_PARSE_COMMAND_PARAMS() here to hook up your commands.
25 // For example:

    ON_PARSE_COMMAND(Default, CSynchronizerExtension, ITS_EMPTY)
    ON_PARSE_COMMAND(BroadcastVideoEvents, CSynchronizerExtension,
ITS_PSTR ITS_PSTR ITS_I4 ITS_PSTR ITS_PSTR ITS_PSTR)
30 ON_PARSE_COMMAND(ServerInfo, CSynchronizerExtension, ITS_PSTR
ITS_PSTR)
    DEFAULT_PARSE_COMMAND(Default, CSynchronizerExtension)
END_PARSE_MAP(CSynchronizerExtension)

35
////////////////////////////////////
// The one and only CSynchronizerExtension object

CSynchronizerExtension theExtension;
40

////////////////////////////////////
// CSynchronizerExtension implementation

45 CSynchronizerExtension::CSynchronizerExtension()
{

```

```

        m_pisapithread = NULL;
    }

    CSynchronizerExtension::~CSynchronizerExtension()
5    {
        delete m_pisapithread;
        m_pisapithread = NULL;
    }
    static HINSTANCE g_hInstance;
10    HINSTANCE AFXISAPI GetResourceHandle()
    {
        return g_hInstance;
    }
15    BOOL CSynchronizerExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
    {
        // Call default implementation for initialization
        CHttpServer::GetExtensionVersion(pVer);
20
        // Load description string
        TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1];
        g_hInstance = AfxGetResourceHandle();
        ISAPIVERIFY(::LoadString(AfxGetResourceHandle(),
25        IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
        _tcscpy(pVer->lpszExtensionDesc, sz);
        BOOL bret = TRUE;
        try
        {
30            m_pisapithread = new isapithread();
            m_pisapithread->start(NULL);
        }
        catch(...)
        {
35            bret = FALSE;
        }
        return bret;
    }

40    BOOL CSynchronizerExtension::TerminateExtension(DWORD dwFlags)
    {
        // extension is being terminated
        //TODO: Clean up any per-instance resources
        if (m_pisapithread)
45            m_pisapithread->stop();
        delete m_pisapithread;
    }

```

```

        m_pisapithread = NULL;
        return TRUE;
    }

5  ///////////////////////////////////////////////////////////////////
    // CSynchronizerExtension command handlers

    void CSynchronizerExtension::Default(CHttpServerContext* pCtxt)
    {
10      StartContent(pCtxt);
        WriteTitle(pCtxt);

        *pCtxt << _T("Invalid function call...");

15      EndContent(pCtxt);
    }

    void CSynchronizerExtension::ServerInfo(CHttpServerContext* pCtxt,
                                           LPCTSTR
20  lpDiskId,
                                           LPCTSTR
                                           nLocationID)
    {
25      StartContent(pCtxt);

        std::string locationId;
        std::string theDiskId = lpDiskId;
        locationId = nLocationID;

30      if (m_pisapithread)
        {
            m_pisapithread->getBLayerInfo(theDiskId,locationId,pCtxt);
        }
        EndContent(pCtxt);

35  }

    void CSynchronizerExtension::BroadcastVideoEvents(CHttpServerContext* pCtxt,
40  LPTSTR lpUserName,
    LPCTSTR lpDiskId,
    long nLocationID,
45  LPCTSTR lpBCA,

```

```

LPCTSTR lpProcessorID,
LPCTSTR lpDecoderID)
5  {
    if (m_pisapithread)
    {
        try
10        {
            m_pisapithread->addrequest(new request(pCtxt->m_pECB,
                const_cast<char *>(lpDiskId),
                nLocationID,
                const_cast<char *>(lpDecoderID),
15                const_cast<char *>(lpUserName),
                const_cast<char *>(lpBCA)));

            pCtxt->m_dwStatusCode = HSE_STATUS_PENDING;
            pCtxt->m_bSendHeaders = FALSE;
20        }
        catch(IAUserException *piauserexc)
        {
            *pCtxt << piauserexc->operator const char *();
            delete piauserexc;
25        }
    }
    else
    {
        *pCtxt << _T("Please comeback later...");
30    }
}

}

35 // Do not edit the following lines, which are needed by ClassWizard.
#ifdef
BEGIN_MESSAGE_MAP(CSynchronizerExtension, CHttpServer)
   //{{AFX_MSG_MAP(CSynchronizerExtension)
   //}}AFX_MSG_MAP
40 END_MESSAGE_MAP()
#endif // 0

45 //////////////////////////////////////
// If your extension will not use MFC, you'll need this code to make

```

```
// sure the extension objects can find the resource handle for the
// module. If you convert your extension to not be dependent on MFC,
// remove the comments around the following AfxGetResourceHandle()
// and DllMain() functions, as well as the g_hInstance global.
5  /****

static HINSTANCE g_hInstance;

10 HINSTANCE AFXISAPI AfxGetResourceHandle()
{
    return g_hInstance;
}

15 BOOL WINAPI DllMain(HINSTANCE hInst, ULONG ulReason,
                      LPVOID lpReserved)
{
    if (ulReason == DLL_PROCESS_ATTACH)
    {
20         g_hInstance = hInst;
    }

    return TRUE;
}
25 ****/
```


ISAPITHREAD.H:

```

// isapithread.h: interface for the isapithread class.
//
5  //////////////////////////////////////

    #if
    !defined(AFX_ISAPITHREAD_H_F39740E6_355C_4103_BD65_2E65078E4D6E_I
    NCLUDED_)
10  #define
    AFX_ISAPITHREAD_H_F39740E6_355C_4103_BD65_2E65078E4D6E__INCLUDE
    D_

    #if _MSC_VER > 1000
15  #pragma once
    #endif // _MSC_VER > 1000

    #include "threadfunc.h"
    #include "layersink.h"
20  #include <map>
    #include <vector>
    class CHTTPServerContext;

    typedef std::map<std::string,layerSink * > mapsink;
25  typedef std::vector<request *> request_vector;
    class isapithread : public threadFunctor
    {
    public:
30      isapithread();
      virtual ~isapithread();

      void addrequest(request *prequest);
      void getBLayerInfo(std::string &diskID,std::string
&locationID,CHttpServerContext* pCtxt);
35  protected:
      void sendTime(CHttpServerContext * pCtxt, long title, long chapter, long
lapsedTime, long eventLength, CTime eventTime, CTime serverTime);
      void processrequest(request *prequest);

40      void handler();
      HRESULT createfactory();

      CComPtr<ILayerFactory> m_ifactory;
      std::string m_dcomserver;
45      mapsink      m_map;
      request_vector m_requestvector;

```

private:

mutex m_mutex;

};

5.

#endif //

!defined(AFX_ISAPITHREAD_H__F39740E6_355C_4103_BD65_2E65078E4D6E__I
NCLUDED_)

10

ISAPITHREAD.CPP:

```

// isapithread.cpp: implementation of the isapithread class.
//
5  //////////////////////////////////////

#include "stdafx.h"
#include "isapithread.h"
#include "cominit.h"
10  #include <comdef.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
15  #define new DEBUG_NEW
#endif

////////////////////////////////////
// Construction/Destruction
20  //////////////////////////////////////

#define THREAD_KEY_T("SOFTWARE\\Interactual Technologies\\ServerConfig")
isapithread::isapithread()
{
25      CRegKey key;
      if (key.Open(HKEY_LOCAL_MACHINE, THREAD_KEY) ==
ERROR_SUCCESS)
      {
30          char value[_MAX_PATH];
          ULONG size = _MAX_PATH;

          ZeroMemory(value, size);
          m_dcomserver.empty();
          try
35          {
              if (key.QueryValue(value, T("DCOMServer"), &size) ==
ERROR_SUCCESS)
              {
40                  m_dcomserver = value;
                  if (m_dcomserver == "(local)")
                      m_dcomserver = "";
              }
          }
          catch(...)
45          {
          }
      }
}

```

```
    }
    else if (key.Create(HKEY_LOCAL_MACHINE, THREAD_KEY) ==
ERROR_SUCCESS)
    {
5         key.SetValue("(local)", T("DCOMServer"));
        m_dcomserver = "";
    }
}

10 isapithread::~isapithread()
{
}

15 void isapithread::handler()
{
    try
    {

20         HRESULT hr = E_FAIL;
        m_mutex.lock();
        // need to create factory;
        hr = createfactory();
25         m_mutex.unlock();

        if (FAILED(hr))
        {
30             throw(new COMException(hr));
        }

        while (1)
        {
            // Copy the request vector so that we don't have to lock the mutex
35         again.

            m_mutex.lock();
            request_vector requestListCopy = m_requestvector;
            m_requestvector.clear();
            m_mutex.unlock();

40             // process each request in the queue
            for (int i = requestListCopy.size() - 1; i >= 0; i--)
            {
45                 processrequest(requestListCopy[i]);
                requestListCopy.pop_back();
            }
        }
    }
}
```

```
        checkCancel();

        m_mutex.lock();
5        // do the processing
        mapsink::iterator it = m_map.begin();
        for (; it != m_map.end(); it++)
        {
            layerSink *pSink = (*it).second;
10            if (pSink && pSink->status() == layerSink::DEAD)
            {
                (*it).second = NULL;
                pSink->clear();
                pSink->Release();
15            }
        }
        m_mutex.unlock();
        Sleep(500);

20    }
    }
    catch(COMException *comException)
    {
        delete comException;
25    }
    catch(...)
    {
    }

30 }

HRESULT isapithread::createfactory()
{
    COSERVERINFO cos ;
35    COAUTHINFO athn;
    WCHAR wsz [MAX_PATH];
    memset(&cos, 0, sizeof(COSERVERINFO));
    HRESULT hr = E_FAIL;
    if (m_dcomserver.size() != 0)
40    {
        MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
        m_dcomserver.c_str(), -1,
        wsz, MAX_PATH);
        cos.pwszName = wsz;
45        cos.pAuthInfo = &athn;
```

```

        athn.pAuthIdentityData = (COAUTHIDENTITY *)new
SEC_WINNT_AUTH_IDENTITY;
        athn.dwAuthnSvc = RPC_C_AUTHN_WINNT;
        athn.dwAuthzSvc = RPC_C_AUTHZ_NONE;
5         athn.pwszServerPrincName = NULL;
        athn.dwAuthnLevel = RPC_C_AUTHN_LEVEL_DEFAULT;
        athn.dwCapabilities = EOAC_NONE;
        athn.pAuthIdentityData->UserLength = _tcslen("dbmgr");
        athn.pAuthIdentityData->DomainLength = _tcslen("IANT_DEVELOP");
10        athn.pAuthIdentityData->PasswordLength = _tcslen("ygm@neo");
        athn.pAuthIdentityData->Domain = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->DomainLength + 1];
        athn.pAuthIdentityData->User = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->UserLength + 1];
15        athn.pAuthIdentityData->Password = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->PasswordLength + 1];
        lstrcpy((TCHAR *)athn.pAuthIdentityData->Domain,
"IANT_DEVELOP");
        lstrcpy((TCHAR *)athn.pAuthIdentityData->User, "dbmgr");
20        lstrcpy((TCHAR *)athn.pAuthIdentityData->Password, "ygm@neo");

    }
    MULTI_QI mqi;

25    mqi.pIID = &IID_ILayerFactory;
    mqi.pItf = NULL;
    mqi.hr = E_FAIL;

    hr = CoCreateInstanceEx(CLSID_LayerFactory, NULL,
30    CLSCTX_ALL, &cos, 1, &mqi);
    char msg[_MAX_PATH];

    sprintf(msg, "hr = %d\n", hr);
    ::OutputDebugString(msg);
35    if (SUCCEEDED(hr))
    {
        ::OutputDebugString("Factory creation succeeded\n");
        hr = mqi.hr;
        if (SUCCEEDED(hr))
40        {
            m_ifactory = reinterpret_cast<ILayerFactory *>(mqi.pItf);
        }
    }

45    return hr;

```

```

    }

void isapithread::addrequest(request *prequest)
{
5     layerSink::sendHeaders(prequest);
    m_mutex.lock();
    m_requestvector.push_back(prequest);
    m_mutex.unlock();
}

10 void isapithread::getBLayerInfo(std::string &diskID,std::string
    &locationID,CHttpServerContext* pCtxt)
{
    mapsink::iterator theIterator;
15     long ldata;
    long title, chapter, eventLength, lapsedTime;
    std::string reqstr = diskID + locationID;
    std::string lEventStart,lEventStop;
    std::string
20     chapterOut,titleOut,startTimeOut,eventLenOut,lapsedTimeOut,serverTimeOut;

    m_mutex.lock();
    theIterator = m_map.find(reqstr);
    layerSink *playerSink = NULL;
25

    if(theIterator != m_map.end() && (*theIterator).second != NULL) // is 0 - 9
    {
        playerSink = (*theIterator).second;
30         IBusinessLayer* m_ilayer;
        m_ilayer = playerSink->getBLayer();
        if (m_ilayer)
        {
            CTime eventTime, serverTime;
35             m_ilayer->get_chapterProperties(&chapter);
            m_ilayer->get_titleProperties(&title);
            m_ilayer->get_startEvent(&ldata);
            eventTime = ldata;
            m_ilayer->get_eventLength(&eventLength);
40             m_ilayer->get_lapsedTime(&lapsedTime);
            m_ilayer->get_serverTime(&ldata);
            serverTime = ldata;

            sendTime(pCtxt,title,chapter,lapsedTime,eventLength,eventTime,serverTime);
45
        }
    }
}

```

```

    }
    else if (m_ifactory)
    {
        long eventTime;
        long serverTime;

        m_ifactory->getScheduledTime(_bstr_t(diskID.c_str()),
        _bstr_t(locationID.c_str()), &eventTime,&serverTime,&eventLength);
        sendTime(pCtxt,0,0,0,eventLength,eventTime,serverTime);
    }
    else
    {
        *pCtxt << _T("_INTERACTUAL_ERROR");
#ifdef _TRACE
        std::string error = "Error occured sending time info: location id = " +
        locationID;
        ::OutputDebugString(error.c_str());
#endif
    }
    m_mutex.unlock();
}

void isapithread::sendTime(CHttpRequestContext *pCtxt, long title, long chapter, long
lapsedTime, long eventLength, CTime eventTime, CTime serverTime)
{
    *pCtxt << (long int)chapter;
    *pCtxt << "\1";
    *pCtxt << (long int)title;
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%S");
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%M");
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%H");
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%d");
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%m");
    *pCtxt << "\1";
    *pCtxt << eventTime.Format("%Y");
    *pCtxt << "\1";
    *pCtxt << (long int)eventLength;
    *pCtxt << "\1";
    *pCtxt << (long int)lapsedTime;
    *pCtxt << "\1";

```



```

        *pCtxt << serverTime.Format("%S");
        *pCtxt << "\1";
        *pCtxt << serverTime.Format("%M");
        *pCtxt << "\1";
5      *pCtxt << serverTime.Format("%H");
        *pCtxt << "\1";
        *pCtxt << serverTime.Format("%d");
        *pCtxt << "\1";
10     *pCtxt << serverTime.Format("%m");
        *pCtxt << "\1";
        *pCtxt << serverTime.Format("%Y");
        *pCtxt << "\1";

    }
15 void isapithread::processrequest(request *prequest)
    {
        mapsink::iterator theIterator;
        std::string reqstr = prequest->get_diskid() + prequest->get_locationid();
20     theIterator = m_map.find(reqstr);
        layerSink *playerSink = NULL;
        try
        {
25     - 9      if(theIterator != m_map.end() && (*theIterator).second != NULL) // is 0
                {
                    playerSink = (*theIterator).second;
                    if (playerSink && playerSink->status() != layerSink::DEAD)
                        playerSink->addRequest(prequest);
30                 else
                    throw new IAUserException("event has ended");
                }
            else
35         {
                playerSink = createLayer();
                playerSink->construct(prequest, m_ifactory, m_dcomserver);
                m_map[reqstr] = playerSink;
            }
40     }
        catch(IAUserException *userException)
        {
            if (playerSink)
                playerSink->Release();
45     playerSink = NULL;
            m_map[reqstr] = 0;
        }
    }

```

```

    std::string msg = userException->operator const char *();
    layerSink::formatdata(msg);
    if (layerSink::writedata(prequest, msg))
        layerSink::closeconnection(prequest);
5      delete prequest;
      delete userException;
  }
10  catch(COMException * comException)
  {
    if (playerSink)
        playerSink->Release();
    playerSink = NULL;
    m_map[reqstr] = 0;
15    delete comException;
    std::string msg;
    layerSink::formatdata(msg);
    if (layerSink::writedata(prequest, msg))
        layerSink::closeconnection(prequest);
20    delete prequest;
  }
  catch(...)
  {
    if (playerSink)
25        playerSink->Release();
    playerSink = NULL;
    m_map[reqstr] = 0;
    std::string msg;
    layerSink::formatdata(msg);
30    if (layerSink::writedata(prequest, msg))
        layerSink::closeconnection(prequest);
    delete prequest;
  }
35 }
```

THREADFUNCTOR.H:

```

// threadImpl.h: interface for the threadImpl class.
//
5  //////////////////////////////////////

    #if
    !defined(AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__
    INCLUDED_)
10  #define
    AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__INCLU
    DED_

    #if _MSC_VER > 1000
15  #pragma once
    #endif // _MSC_VER > 1000
    // main isapi thread
    #include <process.h>
    #include <tchar.h>
20  #include <map>
    #include <vector>
    typedef void (_cdecl * threadFunc)(void *);

    #include "cominit.h"
25  //////////////////////////////////////
    ////////////////////////////////// application event log////////////////////////////////
    class appEventLog
    {
30  public :
        appEventLog(const char * eventName = NULL,
                    const char *srcName = _T("Application"));
        ~appEventLog();

35        bool appReportErrorEvent(const char * msg,
                                WORD type =
                                EVENTLOG_ERROR_TYPE,
                                WORD eventId = 0,
                                WORD category = 7) const;

40  protected:
        HANDLE m_eventLogHandle;
    };

45

```

```

class ExceptionCanceled : public COMException
{
public:
    ExceptionCanceled() : COMException(S_OK)
5      {}

};

////////////////////////////////////
10  ////////////////////////////////// mutex //////////////////////////////////

class mutex
{
public:
15      mutex(LPSECURITY_ATTRIBUTES pattr = NULL, bool bOwned = false, char
* name = NULL)
        { m_mutex = CreateMutex(pattr, bOwned, name); m_locked = false;}
        ~mutex()
        { if (m_mutex) CloseHandle(m_mutex); m_mutex = NULL; }
20      bool isLocked()
        { return m_locked; }
        void lock()
        { ::WaitForSingleObject(m_mutex, INFINITE); m_locked = true;}
        void unlock()
25      {::ReleaseMutex(m_mutex); m_locked = false;}

protected:
        HANDLE m_mutex;
        bool m_locked;
30      };

////////////////////////////////////
35  ////////////////////////////////// threadFunctor //////////////////////////////////

class threadFunctor
{
public:
40      threadFunctor();
        virtual ~threadFunctor();

        void set_event_log(appEventLog * eventlog)
        { m_eventlog = eventlog; }
45      }

```

```

        void writeevent(const char * msg, unsigned int eventId = 0, WORD type =
EVENTLOG_ERROR_TYPE)
        { if (m_eventlog) m_eventlog->appReportErrorEvent(msg, type, eventId); }
        virtual void start(void *pParam, bool bSynch = true);
5  #ifndef _AFX
        private:
        #endif
        virtual void startWin(void *pParam, bool bSynch = true);
        #ifndef _AFX
10 , public:
        #endif

        virtual void signal();
        void stop();
15  HANDLE threadHandle();
        DWORD threadId();

protected:
        void checkCancel();
20  virtual void startThread();

        virtual void prehandler();
        virtual void handler();
        virtual void posthandler();
25

        HANDLE m_threadhandle;
        HANDLE m_hEvent; // event to terminate thread;
        void * m_tParam;
        bool m_sync;
30  appEventLog * m_eventlog;
        DWORD m_dwThreadId;
        friend void _cdecl threadfunction(void *pParam);
        friend UINT mfcthreadfunc( void *pparam );
};
35

////////////////////////////////////
//////////////////////////////////// threadPool //////////////////////////////////////

40  typedef std::map<int, class threadFunctor *> threadMap;

class threadPool
{
public:
45  threadPool();
        virtual ~threadPool();

```

```
public:
    void signal();
    void set_event_log(appEventLog * eventlog);
5    unsigned long wait(unsigned long timeout);
    void execute(int type,void *param);
    void remove(int type);
    void add(int, threadFunctor * pthread);
    void execute();
10    threadFunctor * functor(int type);

protected:
    threadMap    m_threadMap;
15 };

#endif//
20 #ifndef(AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__
    INCLUDED_)
```

THREADFUNCTOR.CPP:

```

// threadImpl.cpp: implementation of the threadImpl class.
//
5  //////////////////////////////////////

#include "stdafx.h"
#include <process.h>
#include "threadfunc.h"
10

#ifdef _AFX
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
15 #define new DEBUG_NEW
#endif // _DEBUG
#endif // _AFX

20  //////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

threadFunctor::threadFunctor()
25 {
    m_tParam = NULL;
    m_threadhandle = NULL;
    m_hEvent = NULL;
    m_dwThreadId = 0;
30     m_sync = false;
    m_eventlog = NULL;
}

threadFunctor::~threadFunctor()
35 {
    if (m_threadhandle)
        CloseHandle(m_threadhandle);
    if (m_hEvent)
        CloseHandle(m_hEvent);
40 }

void _cdecl threadfunction(void *pparam)
{
45     threadFunctor* pfunc = reinterpret_cast<threadFunctor*>(pparam);
    if (pfunc)
    {

```

```

        pfunc->startThread();
    }
    _endthread();
}
5
UINT mfcthreadfunc( void *pparam )
{
    threadFunctor* pfunc = reinterpret_cast<threadFunctor *> (pparam);
    if (pfunc)
10    {
        pfunc->startThread();
    }

    return 0;
15 }

void threadFunctor::startThread()
{
    COMInitializer comInit;
20    m_dwThreadId = ::GetCurrentThreadId();
    if (m_sync)
        ::SetEvent(m_hEvent);

    try
    {
25    prehandler();
        handler();
    posthandler();
    }
    catch(...)
30    {
    }
    m_threadhandle = NULL;
}

35 void threadFunctor::startWin(void *pParam,bool bSynch)
{
#ifdef _AFX
    assert("" == "Tried to invoke threadFunctor::startWin() without MFC");
#else // _AFX
40    m_tParam = pParam;
    if (!m_hEvent)
        m_hEvent = CreateEvent(NULL,false,false, _T("_InterActual_EVENT"));
    m_sync = bSynch;
    m_threadhandle = (HANDLE)AfxBeginThread(mfcthreadfunc,this);
45    if (m_hEvent && bSynch)
    {

```



```
        WaitForSingleObject(m_hEvent, INFINITE);
        ResetEvent(m_hEvent);
    }
    else if (bSynch)
5      {
        throw new IAUserException("Event failed to create");
    }
#endif // _AFX
}
10
void threadFunctor::start(void *pParam, bool bSynch)
{
    m_tParam = pParam;
    m_sync = bSynch;
    if (!m_hEvent)
15      m_hEvent = CreateEvent(NULL, false, false, NULL);
    m_threadhandle = (HANDLE)_beginthread(threadfunction, 0, this);

    if (m_hEvent && bSynch)
20      {
        WaitForSingleObject(m_hEvent, INFINITE);
        ResetEvent(m_hEvent);
    }
    else if (bSynch)
25      {
        throw new IAUserException("Event failed to create");
    }
}

30
void threadFunctor::prehandler()
{
    // Do nothing
}
35

void threadFunctor::handler()
{
    DWORD result;
    while ((result = WaitForSingleObject(m_hEvent, 0)) != WAIT_OBJECT_0)
40      {
        checkCancel();
    }
}
45
```

```
void threadFunctor::posthandler()
{
    // Do nothing
}
5

void threadFunctor::signal()
{
    if (m_hEvent)
10      ::SetEvent(m_hEvent);
}

void threadFunctor::stop()
{
15  #ifndef _WIN32_DCOM
      ATLTRACE("thread %p is stopping", this);
    #endif
    if (m_threadhandle)
    {
20      signal();
      WaitForSingleObject(m_threadhandle, INFINITE);
      if (m_threadhandle)
          CloseHandle(m_threadhandle);
    }
25  CloseHandle(m_hEvent);
    m_threadhandle = NULL;
    m_hEvent = NULL;
}
30

void threadFunctor::checkCancel()
{
35  DWORD result;
    if (m_hEvent)
    {
        result = WaitForSingleObject(m_hEvent, 0);
        switch(result)
40      {
          case WAIT_ABANDONED:
            break;

          case WAIT_OBJECT_0:
45      throw new ExceptionCanceled();
            break;
        }
    }
}
```

```

        };
    }
}

5  HANDLE threadFunctor::threadHandle()
    {
        return m_threadhandle;
    }

10

    DWORD threadFunctor::threadId()
    {
        return m_dwThreadId; // Might not have been set if thread start not synched
15    }

    //////////////////////////////////////
    // threadPool Class
    //////////////////////////////////////

20    //////////////////////////////////////
    // Construction/Destruction
    //////////////////////////////////////

25    threadPool::threadPool()
    {

    }

30    threadPool::~threadPool()
    {
        threadMap::iterator it = m_threadMap.begin();

        for(; it != m_threadMap.end(); it++)
35        {
            threadFunctor * functor = (*it).second;
            if (functor)
            {
                functor->stop();
                delete functor;
40            }
        }
        m_threadMap.clear();
    }

45    void threadPool::add(int type, threadFunctor *pthread)

```

```

    {
        threadMap::iterator it;
        it = m_threadMap.find(type);
        if (it == m_threadMap.end())
5         m_threadMap[type] = pthread;
    }

void threadPool::remove(int type)
{
10     threadMap::iterator it = m_threadMap.find(type);
    if (it != m_threadMap.end() && (*it).second != NULL)
    {
        threadFunctor * functor = (*it).second;
        functor->stop();
15         delete functor;
        m_threadMap[type] = NULL;
    }
}

20 void threadPool::execute(int type, void *param)
{
    threadMap::iterator it = m_threadMap.find(type);
    if (it != m_threadMap.end() && (*it).second != NULL)
25     {
        (*it).second->start(param);
    }
}

30 void threadPool::set_event_log(appEventLog * eventlog)
{
    threadMap::iterator it = m_threadMap.begin();
    for (; it != m_threadMap.end(); it++)
35     {
        if ((*it).second != NULL)
            (*it).second->set_event_log(eventlog);
    }
}

40 void threadPool::execute()
{
    threadMap::iterator it = m_threadMap.begin();
    for (; it != m_threadMap.end(); it++)
45     {

```

```

        threadFunction * functor = (*it).second;
        if (functor != NULL)
        {
            functor->start(functor);
            SleepEx(2, true);
        }
    }
}

threadFunction * threadPool::functor(int type)
{
    threadFunction * functor = NULL;
    threadMap::iterator it = m_threadMap.find(type);
    if (it != m_threadMap.end() && (*it).second != NULL)
    {
        functor = (*it).second;
    }
    return functor;
}

unsigned long threadPool::wait(unsigned long timeout)
{
    unsigned long lRet = -1;
    HANDLE *handleMap = new HANDLE[m_threadMap.size()];
    for (int i = 0; i < m_threadMap.size(); i++)
    {
        handleMap[i] = m_threadMap[i]->threadHandle();
    }
    lRet = ::WaitForMultipleObjects(m_threadMap.size(), handleMap, true, timeout);
    delete handleMap;
    return lRet;
}

void threadPool::signal()
{
    for (int i = 0; i < m_threadMap.size(); i++)
    {
        m_threadMap[i]->signal();
    }
}

////////////////////////////////////

```

```

//////////////////////////////// Application event log //////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////

5  const char *SubKey =
    _T("System\\CurrentControlSet\\Services\\EventLog\\Application\\");

appEventLog::appEventLog(const char * eventName, const char * srcName)
{
10      USES_CONVERSION;
        m_eventLogHandle = NULL;
        std::string subKey = std::string(SubKey) + srcName;
        CRegKey      regkey;

15      regkey.Create(HKEY_LOCAL_MACHINE, subKey.c_str());

        m_eventLogHandle = RegisterEventSource(eventName, srcName);

        if (!m_eventLogHandle)
20      {
                ATLTRACE("Could not register event source error = %d",
                GetLastError());
        }
}

25 appEventLog::~appEventLog()
{
        if (m_eventLogHandle)
                DeregisterEventSource(m_eventLogHandle);
30 }

bool appEventLog::appReportErrorEvent(const char * msg,
        WORD type /*= EVENTLOG_ERROR_TYPE*/,
        WORD eventId /*= 0*/,
35      WORD category /*= 7*/) const
{
        bool bresult = false;
        if (m_eventLogHandle)
        {
40      #pragma warning(push)
        #pragma warning(disable : 4800)
                bresult = ReportEvent(m_eventLogHandle, type, category, eventId,
                NULL,
                        1, 0, (const char **)&msg, NULL);
45      #pragma warning(pop)
        }
}

```

```
    return bresult;  
}
```

BUSINESSLAYER.H:

```

// BusinessLayer.h : Declaration of the CBusinessLayer

5  #ifndef __BUSINESSLAYER_H_
    #define __BUSINESSLAYER_H_

        #include "resource.h"    // main symbols

10  #include "ConfigurationManager.h"
    #include "DBConnector.h"
    #include <comdef.h>

    #include <map>

15  #include <string>
    #include "HiddenWnd.h"
    #include "LayerImplCP.h"

20  #include "threadfunc.h"

    class layerthread;

    typedef std::map<long,long> decoderCapabilities;
25  //////////////////////////////////////
    // CBusinessLayer
    class ATL_NO_VTABLE CBusinessLayer :
        public CComObjectRootEx<CComMultiThreadModel>,
        public CComCoClass<CBusinessLayer, &CLSID_BusinessLayer>,
30  public ISupportErrorInfo,
        public IConnectionPointContainerImpl<CBusinessLayer>,
        public IBusinessLayer,
        public CProxy_IBusinessLayerEvents< CBusinessLayer >
    {
35  protected:
        std::string m_diskid;
        std::string m_location;
        _bstr_t m_diskID;
        _bstr_t m_locationID;
40  CTime m_serverTime;
        CTime m_eventLength;
        CTime m_lapsedTime;
        CTime m_startEvent;
        CTime m_stopEvent;
45  long threshold;
        short m_hostType;

```



```

    long m_title;
    long m_chapter;
    IDB_Connector *m_pIDBConnect;
    ICCConfigMgrImpl *m_pICCConfigMgrImpl;
5    layerthread * m_pthread;
    bool m_firstTime;
    decoderCapabilities m_capabilities;

10    protected:
        void ChkValidEvent();
        HRESULT GetNextPair(long *theTime, long *nTitle, long * nChapter, BSTR
        *chapterCmnd);
        void sendCommand(BSTR chapterCmnd);
15    void endSession(BSTR szMsg);
        void updateTime(LONG time, LONG nTitle);
        _bstr_t firstdvdCmd();
        void put_serverTime(/*[in]*/ long serverTime, long title, long chapter, long
        lapsedTime, long length);
20    mutex m_timeLock;

    public:
        CBusinessLayer()
        {
25            m_pthread = NULL;
            m_pIDBConnect = NULL;
            m_pICCConfigMgrImpl = NULL;
            m_diskid.empty();
            m_location.empty();
30        }

        HRESULT FinalConstruct();
        void FinalRelease();

35    DECLARE_REGISTRY_RESOURCEID(IDR_BUSINESSLAYER)

    DECLARE_PROTECT_FINAL_CONSTRUCT()

    BEGIN_COM_MAP(CBusinessLayer)
40        COM_INTERFACE_ENTRY(IBusinessLayer)
        COM_INTERFACE_ENTRY(ISupportErrorInfo)
        COM_INTERFACE_ENTRY(IConnectionPointContainer)
        COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
    END_COM_MAP()
45    BEGIN_CONNECTION_POINT_MAP(CBusinessLayer)

```

```

CONNECTION_POINT_ENTRY(IID_IBusinessLayerEvents)
END_CONNECTION_POINT_MAP()

5

// ISupportsErrorInfo
    STDMETHODCALLTYPE(InterfaceSupportsErrorInfo)(REFIID riid);

10 // IBusinessLayer
    void ProcessEvent();

    public:
        STDMETHODCALLTYPE(get_serverTime)(/*[out, retval]*/ long *pVal);
15        STDMETHODCALLTYPE(get_titleProperties)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(get_chapterProperties)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(get_lapsedTime)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(get_eventLength)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(TranslateTimePlay)(/*[in]*/ long nDecoderType, /*[in]*/ long
20        nTitle, /*[in]*/ long nTime, /*[out, retval]*/ BSTR *szCmd);
        STDMETHODCALLTYPE(Initialize());
        STDMETHODCALLTYPE(get_threshold)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(put_threshold)(/*[in]*/ long newVal);
        STDMETHODCALLTYPE(get_stopEvent)(/*[out, retval]*/ long *pVal);
25        STDMETHODCALLTYPE(put_stopEvent)(/*[in]*/ long newVal);
        STDMETHODCALLTYPE(get_startEvent)(/*[out, retval]*/ long *pVal);
        STDMETHODCALLTYPE(put_startEvent)(/*[in]*/ long newVal);
        STDMETHODCALLTYPE(get_location)(/*[out, retval]*/ BSTR *pVal);
        STDMETHODCALLTYPE(put_location)(/*[in]*/ BSTR newVal);
30        STDMETHODCALLTYPE(get_disk)(/*[out, retval]*/ BSTR *pVal);
        STDMETHODCALLTYPE(put_disk)(/*[in]*/ BSTR newVal);
    friend class layerthread;
};

35 #endif // __BUSINESSLAYER_H_

```

BUSINESSLAYER.CPP:

```

// BusinessLayer.cpp : Implementation of CBusinessLayer
#include "stdafx.h"
5  #include "layerimpl.h"
    #include "BusinessLayer.h"
    #include "process.h"

10  class layerthread : public threadFunctor
    {
    public:
        void startThread()
        {
            COMInitializer init;
15             try
            {
                handler();
            }
            catch(...)
20             {
            }
            m_threadhandle = NULL;
        }
        void handler()
25         {
            long threshold;
            long nextChapterTime;
            CTime startEvent;
            CTime stopEvent;
30             CTime time;
            long eventLength;
            char msg[256];

            ::OutputDebugString("\n Loop started\n");
35             CBusinessLayer *pLayer = reinterpret_cast<CBusinessLayer
*>(m_tParam);
            if (pLayer)
            {
                try
40                 {

                    time = CTime::GetCurrentTime() ;
                    pLayer->get_threshold(&threshold);
                    long timeData;
45                     pLayer->get_startEvent(&timeData);
                    startEvent = timeData;

```

```

pLayer->get_stopEvent(&timeData);
stopEvent = timeData;

eventLength = CTimeSpan(stopEvent -
5  startEvent).GetTotalSeconds() * 1000;

sprintf(msg, "%s\n", startEvent.Format("%D:%H:%M"));
::OutputDebugString(msg);
sprintf(msg, "%s\n", stopEvent.Format("%D:%H:%M"));
10  ::OutputDebugString(msg);
pLayer->put_serverTime(time.GetTime(), -1, -1, 0,
eventLength);

if (startEvent < stopEvent &&
15  (time + CTimeSpan(threshold)) >= startEvent)
{
//
// - Determine if it is time to kick-off the event.
// - If it is, stop the loop.
// - If it is NOT, go to sleep and check after 500
20  milliseconds

//
_bstr_t dvdCmd = pLayer->firstdvdCmd();
std::string debugmsg;
debugmsg = "first dvd command" + dvdCmd + "\n";
25

::OutputDebugString(debugmsg.c_str());
while (time < startEvent )
{
30  checkCancel();
Sleep(500);
time = CTime::GetCurrentTime() ;
pLayer->put_serverTime(time.GetTime(), -1,
-1, 0, eventLength);
}
35  if (time <= (startEvent + CTimeSpan(1)))
{
pLayer->sendCommand(dvdCmd);

::OutputDebugString("Process Event");
40  }

while (time < stopEvent)
{
45  BSTR command;
CTimeSpan lapsedTime(0);

```

```

long title, chapter;

if (time > startEvent)
{
    5         lapsedTime = time - startEvent;
}

if (SUCCEEDED(pLayer-
>GetNextPair(&nextChapterTime,&title, &chapter, &command)))
10     {
        if (lapsedTime.GetTotalSeconds() <
nextChapterTime)
        {
15             while(lapsedTime.GetTotalSeconds() <= nextChapterTime)
                {
                    checkCancel();
                    Sleep(500);
                    time =

20             CTime::GetCurrentTime();

                    lapsedTime = time -
startEvent;
                    pLayer-
25             >updateTime(lapsedTime.GetTotalSeconds(),title);
                    pLayer-
>put_serverTime(time.GetTime(),title, chapter ,lapsedTime.GetTotalSeconds() * 1000,
eventLength);
                }
30             pLayer-
>sendCommand(command);
                    Sleep(500);
                }
35             time = CTime::GetCurrentTime() ;
::SysFreeString(command);
        }
        else
        {
40             TRACE("no more entries in the
chapter table");
                    break;
        }
    }
45     }
}

```

```

        ::OutputDebugString("End Session");
        pLayer->endSession(NULL);
    }
    catch(ExceptionCanceled * pExcp)
    {
        delete pExcp;
        _bstr_t msg = "Cancellation occurred";
        pLayer->endSession(msg);
    }
    catch(...)
    {
        _bstr_t msg = "Cancellation occurred";
        pLayer->endSession(msg);
    }
}

};

////////////////////////////////////
// CBusinessLayer

25  STDMETHODCALLTYPE CBusinessLayer::InterfaceSupportsErrorInfo(REFIID riid)
    {
        static const IID* arr[] =
        {
            &IID_IBusinessLayer
30  };
        for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
        {
            if (InlineIsEqualGUID(*arr[i],riid))
                return S_OK;
35  }
        return S_FALSE;
    }

HRESULT CBusinessLayer::FinalConstruct()
40  {

        HRESULT hr = CoCreateInstance(CLSID_CConfigMgrImpl,
                                     0,
45  CLSCTX_ALL,
                                     IID_ICConfigMgrImpl,

```

```

        (void**)&m_pICConfigMgrImpl);

        return hr;
5    }

void CBusinessLayer::FinalRelease()
{
    if (m_pthread)
10        m_pthread->stop();
    delete m_pthread;
    if (m_pICConfigMgrImpl)
        m_pICConfigMgrImpl->Release();
    try
15    {
        if (m_pIDBConnect)
            m_pIDBConnect->Release();
    }
    catch(...)
20    {
    }
}

void CBusinessLayer::ChkValidEvent()
25    {
        ::OutputDebugString("\n Check valid Event\n");
        m_firstTime = false;
        if (m_diskID.length() == 0)

30            throw new IAUserException("Invalid Disk id");

        if (m_pICConfigMgrImpl)
        {
            std::string debugmsg;
            debugmsg = "disk id=" + m_diskID + "; location id = " + m_locationID +
35            "\n";
            ::OutputDebugString(debugmsg.c_str());
            m_pICConfigMgrImpl->put_diskID(m_diskID);           // Variable
used for search criteria
40            m_pICConfigMgrImpl->put_locationID(m_locationID); // Variable used
for search criteria
            m_pICConfigMgrImpl->get_hostType(&m_hostType);

            if (m_hostType)
45            {
                //

```

```

// Create a DBConnector, store the pointer for future use.
// Store values from db.
//
5      ::OutputDebugString("\n Host type is checked\n");
      HRESULT hr = S_OK;
      if (!m_pIDBConnect)
      {
          hr = CoCreateInstance(CLSID_DB_Connector,
10                                     0,
                                     CLSCTX_ALL,
                                     IID_IDB_Connector,
                                     (void**)&m_pIDBConnect);
      }
15      if (SUCCEEDED(hr))
      {
          ::OutputDebugString("\n Initialize DB Connector\n");
          m_pIDBConnect->put_diskID(m_diskID); //
          Variable used for search criteria
20      m_pIDBConnect->put_locationID(m_locationID); //
          Variable used for search criteria
          m_pIDBConnect->chkEvent();
          BSTR data;
          m_pIDBConnect->get_diskID(&data);
25      if (data)
          {
              m_diskID = data;
              ::SysFreeString(data);
          }
          m_pIDBConnect->get_locationID(&data);
          if (data)
          {
30              m_locationID = data;
              ::SysFreeString(data);
          }
          long time;
          m_pIDBConnect->get_startEvent(&time);
          m_startEvent = time;
          m_pIDBConnect->get_stopEvent(&time);
40      m_stopEvent = time;
          m_pIDBConnect->get_thresold(&threshold);
          m_pIDBConnect->get_hostType(&m_hostType);

          long * nDecoderArray;
          long * nCapabilitiesArray ;
          nDecoderArray = nCapabilitiesArray = NULL;
45

```



```

        if (SUCCEEDED(m_pIDBConnect-
>decoderArray(&nDecoderArray, &nCapabilitiesArray)))
        {
5           int i = 0;
           while(nDecoderArray[i] != -1)
           {
               m_capabilities[nDecoderArray[i]] =
10          nCapabilitiesArray[i];
               i++;
           }
           CoTaskMemFree(nDecoderArray);
           CoTaskMemFree(nCapabilitiesArray);
        }

15         ::OutputDebugString("\n Prepare to start thread\n");

        m_pthread = new layerthread;
        m_pthread->start(this,false);
20     }
    else
    {
        throw new COMException(hr);
    }

25 }
else
{
    //
30     // Create a Reference Connector, and store the pointer for future
    use.
        // TBD
        //
35 }
}

STDMETHODIMP CBusinessLayer::get_disk(BSTR* pVal)
40 {
    // TODO: Add your implementation code here
    *pVal = m_diskID.copy( );
    return S_OK;
}

45 STDMETHODIMP CBusinessLayer::put_disk(BSTR newVal)

```

```

    {
        // TODO: Add your implementation code here
        m_diskID = newVal;
        return S_OK;
5    }

STDMETHODIMP CBusinessLayer::get_location(BSTR* pVal)
{
    // TODO: Add your implementation code here
10    *pVal = m_diskID.copy( );
    return S_OK;
}

STDMETHODIMP CBusinessLayer::put_location(BSTR newVal)
15 {
    // TODO: Add your implementation code here
    m_locationID = newVal;

    return S_OK;
20 }

STDMETHODIMP CBusinessLayer::get_startEvent(long *pVal)
{
    // TODO: Add your implementation code here
25    *pVal = m_startEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
    return S_OK;
}

STDMETHODIMP CBusinessLayer::put_startEvent(long newVal)
30 {
    // TODO: Add your implementation code here
    time_t time = newVal;
    m_startEvent = time;
    return S_OK;
35 }

STDMETHODIMP CBusinessLayer::get_stopEvent(long *pVal)
{
    // TODO: Add your implementation code here
40    *pVal = m_stopEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
    return S_OK;
}

STDMETHODIMP CBusinessLayer::put_stopEvent(long newVal)
45 {
    // TODO: Add your implementation code here

```

```

        time_t time = newVal;
        m_stopEvent = time;
        return S_OK;
    }
5  STDMETHODCALLTYPE CBusinessLayer::get_threshold(long *pVal)
    {
        // TODO: Add your implementation code here
        *pVal = threshold; // m_pIDBConnect->get_threshold(pVal);
10     return S_OK;
    }

    STDMETHODCALLTYPE CBusinessLayer::put_threshold(long newVal)
    {
15     // TODO: Add your implementation code here
        threshold = newVal;
        return S_OK;
    }

20     HRESULT CBusinessLayer::GetNextPair(long *theTime, long *nTitle, long * nChapter,
        BSTR *chapterCmd)
    {
        return m_pIDBConnect-
>get_NextChapter(theTime,nTitle,nChapter,chapterCmd);
25     }

30     _bstr_t CBusinessLayer::firstdvdCmd()
    {
        //
        // Execute the first DVD Command
        //
35     BSTR msg = NULL;
        _bstr_t dvdMsg;
        m_pIDBConnect->get_initialDVDCommand(&msg);
        if (msg)
            dvdMsg = msg;
40     return dvdMsg;
    }

    void CBusinessLayer::sendCommand(BSTR szMsg)
    {
45     Fire_sendCommand(szMsg);
    }

```

```
    }

    void CBusinessLayer::endSession(BSTR szMsg)
    {
5        Fire_endSession(szMsg);
    }

10 void CBusinessLayer::updateTime(LONG time, long nTitle)
    {
        Fire_updatetime(time,nTitle);
    }

15 STDMETHODCALLTYPE CBusinessLayer::Initialize()
    {
        HRESULT hr = S_OK;
        try
        {
20            ChkValidEvent();
        }
        catch(IAUserException *pexcpt)
        {
            delete pexcpt;
25            hr = E_FAIL;
            _bstr_t msg = "USER exception occurred\n";
            Fire_endSession(msg);
        }
        catch(COMException * pcomexcpt)
30        {
            hr = pcomexcpt->operator HRESULT();
            _bstr_t msg = "COM exception occurred\n";
            Fire_endSession(msg);
        }
35        catch(...)
        {
            _bstr_t msg = "Unknown exception occurred\n";
            Fire_endSession(msg);
40        }

        return hr;
    }

45 STDMETHODCALLTYPE CBusinessLayer::TranslateTimePlay(long nDecoderType, long
nTitle, long nTime, BSTR *szCmd)
```

```
{
    // TODO: Add your implementation code here
    HRESULT hr = E_FAIL;
    decoderCapabilities::iterator it = m_capabilities.find(nDecoderType);
5    if (it != m_capabilities.end())
    {
        if ((*it).second == 0)
        {
            char translate[_MAX_PATH];
            sprintf(translate, "tmp:%d:%d", nTitle, nTime * 1000);
            *szCmd = _bstr_t(translate).copy();
            hr = S_OK;
        }
    }
15    }

    return hr;
}

20 STDMETHODCALLTYPE CBusinessLayer::get_eventLength(long *pVal)
{
    // TODO: Add your implementation code here
    m_timeLock.lock();
    *pVal = m_eventLength.GetTime();
25    m_timeLock.unlock();
    return S_OK;
}

30 STDMETHODCALLTYPE CBusinessLayer::get_lapsedTime(long *pVal)
{
    // TODO: Add your implementation code here
    m_timeLock.lock();
    *pVal = m_lapsedTime.GetTime();
35    m_timeLock.unlock();

    return S_OK;
}

40 STDMETHODCALLTYPE CBusinessLayer::get_chapterProperties(long *pVal)
{
    // TODO: Add your implementation code here
    m_timeLock.lock();
45    *pVal = m_chapter;
    m_timeLock.unlock();
}
```

```

        return S_OK;
    }

5  STDMETHODCALLTYPE CBusinessLayer::get_titleProperties(long *pVal)
    {
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_title;
10     m_timeLock.unlock();
        return S_OK;
    }

15 STDMETHODCALLTYPE CBusinessLayer::get_serverTime(long *pVal)
    {
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_serverTime.GetTime();
20     if (*pVal == 0)
        {
            char msg[1024];

            sprintf(msg, "title = %d, chapter = %d, location = %s\n", m_title,
25     m_chapter, m_locationID.operator char *());
        }
        m_timeLock.unlock();

        return S_OK;
30 }
void CBusinessLayer::put_serverTime(/*[in]*/ long serverTime, long title, long chapter,
long lapsedTime, long length)
{
    m_timeLock.lock();
35     m_serverTime = serverTime;
    m_title = title;
    m_chapter = chapter;
    m_lapsedTime = lapsedTime;
    m_eventLength = length;
40     m_timeLock.unlock();
}

```